REDUCE SCANNING TIME INCREMENTAL ALGORITHM (RSTIA) OF ASSOCIATION RULES

Yehia.M EL-Rahman
Department of Computer Science,
Philadelphia University, Amman,
JORDAN
Yehia_by@hotmail.com

Mohammad.M AL-Widyan
Department of Computer Science,
Al al-Bayt University, Mafraq,
JORDAN
Mohamdwidyan@hotmail.com

ABSTRACT

In the real world where large amounts of data grow steadily, some old association rules can become stale, and new databases may give rise to some implicitly valid patterns or rules. Hence, updating rules or patterns is also important. A simple method for solving the updating problem is to reapply the mining algorithm to the entire database, but this approach is time-consuming. This paper reuses information from old frequent itemsets to improve its performance and addresses the problem of high cost access to incremental databases in which data are very changing by reducing the number of scanning times for the original database. a log file has been used to keep track of database changes Whenever, a transaction has been added, deleted or even modified, a new record is added to the log file. This helps identifying the newly changes or updates in incremental databases. A new vertical mining technique has been used to minimize the number of scanning times to the original database. This algorithm has been implemented and developed using C#.net and applied to real data and gave a good result comparing with pure Apriori.

Keywords: Data mining; Vertical mining; Association rules; Incremental databases.

INTRODUCTION

Recent advances in data mining have attracted much attention in database research. And this is because their wide applicability in many areas, including industry and the finance sector. The availability of automated tools has enabled the collection of large amount of data. These large databases contain information that is potentially useful for making market analysis and financial forecasts.

So, data mining is an approach to discover such useful information from very large and dynamic databases. This information includes association rules, characteristic rules, classification rules, generalized relations, discriminant rules, etc.

There are various data mining problems, but the mining of association rules is an important one. A well-known example for association rules is about basket market analysis . where a record in the sales data describes all the items that are bought in a single transaction. Together with other information such as the transaction time, customer-id, etc. mining association rules from such a database is to discover from the huge amount of past transactions , all the rules like "A customer who buys item A and item B is most likely to buy item C in the same transaction". Where A,B and C are initially unknown. Such rules are very useful for marketers to develop and implement customized marketing programs and strategies.

A feature of data mining problems is that in order to have stable and reliable results, a huge amount of data has to be collected and analyzed. The large amount of input data and mining results poses a maintenance problem. While new transactions are being appended to a database and obsolete ones are being removed. Rules that already discovered also have to be updated. In this paper we examine the problem of maintaining discovered association rules. We propose a new incremental algorithm which can handle all the update cases including insertion, deletion and modification of transactions.

STATEMENT OF PROBLEM

In Apriori (Agrawal, et al., 1993), the discovery of frequent itemsets from transactional databases is accomplished in a step wise fashion, where itemsets found to be frequent in a particular step (n) are used to produce potential frequent itemsets, known as candidate itemsets, at step n+1. During each step, a database scan is essential to perform support counting of the new candidate itemsets. After the presentation of Apriori, numerous association rule algorithms (Zaki, et al., 1997; Han, et al., 2001) have focused on improving Apriori candidate generation step by reducing the number of database passes , main memory usage and other CPU costs.

In this paper, a new incremental technique based on vertical mining has been introduced which aims to reduce the number of scanning time to the original database and to update the stale patterns or rules without reapplying the mining algorithm to the entire database which is time-consuming.

LITERATURE REVIEW

In the real world where large amounts of data grow steadily, some old association rules can become stale, and new databases may give rise to some implicitly valid patterns or rules. Hence, updating rules or patterns is also important. A simple method for solving the updating problem is to reapply the mining algorithm to the entire database, but this approach is time-consuming. The algorithm in this paper reuses information from old frequent itemsets to improve its performance. Several other approaches to incremental mining have been proposed.

Although many mining techniques for discovering frequent itemsets and associations have been presented, the process of updating frequent itemsets remains trouble for incremental databases. The mining of incremental databases is more complicated than the mining of static transaction databases, and may lead to some severe problems, such as the combination of frequent itemsets occurrence counts in the original database with the new transaction database, or the rescanning of the original database to check whether the itemsets remain frequent while new transactions are added.
This work proposes an algorithm for incremental mining, which can discover the latest rules and doesn't need to rescan the original database.

In (D. W. Cheung) the authors have proposed an algorithm called Fast Update algorithm (FUP) to efficiently generate associations in the updated database. The FUP algorithm relies on Apriori and considers only these newly added transactions. Let db be a set of new transactions and DB be the updated database (including all transactions of DB and db). An itemset x is either frequent or infrequent in DB or db. Therefore, x has four possibilities, as shown in table 1. in the first pass, FUP scans db to obtain the occurrence count of each 1-itemset. Since the occurrence counts of Fk in DB are known in advance, the total occurrence count of arbitrary x is easily calculated if x is in Case 2. if x is unfortunately in Case 3, DB must be rescanned. Similarly, the next pass scans db to count the candidate 2-itemsets of db. If necessary, DB is rescanned. The process is reiterated until all frequent itemsets have been found. In the worst case, FUP does not reduce the number of the original database must be scanned.

Table 1. Four scenarios associated with an itemset in DB

db DB	Frequent itemset	Infrequent itemset
Frequent itemset	Case 1: Frequent	Case 2:
Infrequent itemset	Case 3:	Case 4: Infrequent

Furthermore, a recently proposed associative algorithm called MCAR (Thabatah, et al., 2004) adopts the tid-list intersection methods of (Zaki et al., 1997) from association rule to discover classification rules in a single training data scan. A tid-list of an item is the transaction numbers (tids) in the database that contain that item. Experimental results on real world data and synthetic data (Thabatah, et al., 2004) revealed that algorithms that employ tid-lists fast intersection method outperform Apriori-like ones with regards to processing time and memory usage. In spite of the advantage of tid-list intersection approach, when the cardinality of the tid-list becomes very large, intersection time gets larger as well. This happens particularly for large and correlated transactional databases.

Finally, developing efficient frequent ruleitems discovery methods, which decrease the number of database scans and minimize the use of complex data structure objects during the learning step is vital. This is because that most of the time is spent during the training phase.

Apriori Algorithm

The Apriori algorithm concentrates primarily on the discovery of frequent itemsets according to a user-defined minSup. The algorithm relies on the fact that an itemset could be frequent only when each of its subset is frequent; otherwise, the itemset is infrequent. In the first pass, the Apriori algorithm constructs and counts all 1-itemsets. (A k-itemset is an itemset that includes k items.) After it has found all frequent 1-itemsets, the algorithm joins the frequent 1-itemsets with each other to form candidate 2-itemsets. Apriori scans the transaction database and counts the candidate 2-itemsets to determine which of the 2-itemsets are frequent. The other passes are made accordingly. Frequent (k - 1)-itemsets are joined to form k-itemsets whose first k-1 items are identical. If k 3, Apriori prunes some of the k-itemsets; of these, (k – 1)-itemsets have at least one infrequent subset. All remaining k-itemsets constitute candidate k-itemsets. The process is reiterated until no more candidates can be generated.

Example 1: Consider the database presented in Table 2 with a minimum support requirement is 50%. The database includes 11 transactions. Accordingly, the supports of the frequent itemsets are at least six. The first column "TID" includes the unique identifier of each transaction, and the "Items" column lists the set of items of each transaction.

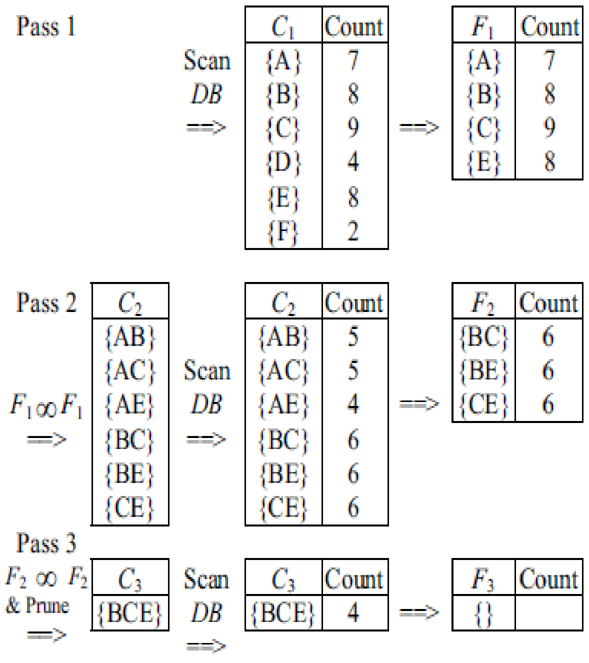
Let Ck be the set of candidate k-itemsets and Fk be the set of frequent k-itemsets. In the first pass, the database is scanned to count C1. If the support count of a candidate exceeds or equals six, then the candidate is added to F1. The outcome is shown in Figure 1. Then, F1 & F1 forms C2 (Apriori-gen function is used to generate C2) after the database has been scanned for a second time, Apriori examines which itemset of C2 exceeds the predetermined threshold.

Moreover, C3 is generated from F2 as follows. Figure 1 presents two frequent 2-itemsets with identical first item, such as {BC} and {BE}. Then, Apriori tests whether the 2-itemset {CE} is frequent. Since {CE} is a frequent itemset, all the subsets of {BCE} are frequent. Thus, {BCE} is a candidate 3-itemset, or {BCE} must be pruned. Apriori stops to look for frequent itemsets when no candidate 4-itemset can be joined from F3. Apriori scans the database k times when candidate k-itemsets are generated.

Table 2: An example of a transaction database

TID	Items
001	A C D E
002	A C D
003	B C E
004	A B C E
005	A B E
006	B C E
007	A B E
008	B C D E
009	A B C D
010	C E F
011	A B C F

Figure 1: application of Apriori algorithm



Reduce Scanning Time Incremental Algorithm (RSTIA)

Apriori algorithm is based on finding large itemsets from database transactions by keeping a count for every itemset. However, since the number of possible itemsets is exponential to the number of items in the database, it is impractical to count every subset we encounter in the database transactions. The Apriori algorithm tackles the combinatorial explosion problem by using an iterative approach to count the itemsets.

The iterative nature of the Apriori algorithm implies that at least n database passes are needed to discover all the large itemsets if the biggest large itemsets are of size n. since database passes involve slow access, to increase efficiency, we should minimize the number of database passes during the mining process. one solution is to generate bigger-sized candidate itemsets as soon as possible, so that their supports can be counted early.

```
foreech(tranID in DelT)
for(j=0 ; j< Ln.Count ; j++) // n = 1……….
if (tranID Exist in Ln[j]. TransactionList)
{
Ln[j].TransactionList.remove(tranID);
Ln[j].Support--;
}

$L1$ ← pass(AddT);
**for** ($k$ = 2; $F_{k-1} \neq \emptyset$; $k$++) **do**
 $L_k$ ← candidate-gen($L_{k-1}$);
 **for** each transaction $t \in$ AddT **do**
 **for** each candidate $l \in L_k$ **do**
**if** $l$ is contained in $t$ **then**
```

```
{ l.Support++;
l.TransactionList.Add(t) }
 end
end
end

Lf ← {$L \in L_k$ | $L.count/n \geq minsup$}
generateRule(Lf);

save(L1,Ln,Lf,Rule);
```

Figure 2: RSTIA Algorithm

This paper addresses the problem of high cost access to incremental databases in which data are very changing and time-varying by reducing the number of scanning times for the original database.

In this approach, a log file has been used to keep track of database changes, the log file contains three columns which are the transactionID, ActionID and finally ActionDate. Whenever, a transaction has been added, deleted or even modified, a new record is added to the log file. This helps identifying the newly changes or updates in incremental databases by avoiding scanning database to locate newly updates. On the other hand, an xml file has been used to store the date of the last time the algorithm has been executed to ensure that on the next time the algorithm should be executed from the last time it executed.

How It Works

This algorithm is Apriori-based but, tries to solve the shortcoming of multi scan to the original database by generating bigger-sized candidate itemsets as soon as possible, so that their supports can be counted early.

And this can be summarized as follows:

Step 1: check the xml file, if it exists then go step 2, else call vertical_ Apriori.

Step 2: read the log file and compare the ActionDate attribute values with the stored value in the xml file, if they are greater than the value of xml file, then go to step 3

Step 3: for each record in the log file that satisfy step 2, check the value of ActionID, if one then go to step 4, if two got step5, if 3 go to steps 5,4 respectivly.

Step 4: The value one of ActionID means a new transaction has been added to the database, so, for each item that involved in such transaction, increment its support by one, and concatenate the transactionID to each item involved in it.

Step 5: The value two of ActionID a transaction has been deleted from the database, so, for each item that involved in such transaction, decrement its support by one, and remove the transactionID from each item involved in it.

VERTICAL_ APRIORI:

it is the same for normal Apriori, but the only two differences are, that a new information is added to frequent times which is the transactions where they lay, and the other difference is that vertical_Apriori scans database first time only and store all frequent tiems , their support, and the transactions they lay in a text file, so in the next time the algorithm run, it no longer scans the

ISSN: 2223-9553
Volume 1, Issue 2, September 2011

database, rather it check the text files which number of passes on the database for future times the algorithm runs.

Example 1: Consider the database presented in Table 3 with a minimum support requirement is 50% and xml file doesn't exists. The database includes 4 transactions. Accordingly, the supports of the frequent itemsets are at least 2. The first column "TID" includes the unique identifier of each transaction, and the "Item" column lists the set of items of each transaction.

Table 3: An example of a transaction database

TID	Item
1	Ab
2	Bc
3	Ac
4	Abc

LOG File ⟹

TID	ActionID	Date
1	1	2011/05/18 12:30
2	1	2011/05/18 12:35
3	1	2011/05/18 12:40
4	1	2011/05/18 12:42

First, there is no rule-generation has been done yet so the algorithm scans the original database and counts all frequent 1-itemsets and store them into FrequentItemsL1 file with their support and transactionID they located to make it easily finding such transactions, see table below

ListIndex	Itemset	Support	TransactionList
1	A	3	1,3,4
2	B	3	1,2,4
3	C	3	2,3,4

In this step, we use FrequentItemsL1 file to generate Candidate itemset of size 2, C2 and count their support by scanning the original database and store them to AllFrequentItem file as it is in previous step

ListIndex	Itemset	Support	TransactionList
1	Ab	2	1,4
2	Ac	2	3,4
3	Bc	2	2,4

In the next step, C3 is generated and their support is counted again by scanning the original database and then the resultant C3 with support are appended to the AllFrequentitmem file.

istIndex	Itemset	Support	TransactionList
1	ab	2	1,4
2	ac	2	3,4
3	bc	2	2,4
1	abc	1	4

In this step, Aproiri Stops, since there is no more candidate itemsets, so we use the AllFrequentitem file to generate The final all frequent item set by matching the required support with their support, and discover the most frequent itemsets in the database.

ListIndex	Itemset	Support
1	a	3

2	b	3
3	c	3
4	ab	2
5	ac	2
6	bc	2

And finally, the rules are generated as follows:

ListIndex	Itemset	confidence
1	a->b	66.6%
2	a->c	66.6%
3	b->a	66.6%
4	b->c	66.6%
5	c->a	66.6%
6	c->b	66.6%

CONCLUSION

This paper reuses information from old frequent itemsets to improve its performance and addresses the problem of high cost access to incremental databases in which data are very changing by reducing the number of scanning times for the original database. a log file has been used to keep track of database changes.

Whenever, a transaction has been added, deleted or even modified, a new record is added to the log file. This helps identifying the newly changes or updates in incremental databases. This algorithm has been implemented and developed using C#.net and applied to real data and gave a good result comparing with pure Apriori.

ACKNOWLEDGEMENTS

Authors would like to thank Dr. Hasan AL-Refai from Philadelphia University for his supported and help. This support is gratefully acknowledged.

REFERENCES

1. Arawal, R., Amielinski, T., and Swami, A. (1993) Mining association rule between sets of items in large databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, (pp. 207-216). Washington, DC.

2. D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, “Maintenance of discovered association rules in large databases: an incremental updating technique,” In Proc. 12th Intl. Conf. on Data Engineering, New Orleans, LA, pp. 106-114, Feb. 1996.

3. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. 3rd KDD Conference, pp. 283-286, August 1997.

4. W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple-class association rule. In ICDM’01, pp. 369-376, San Jose, CA, Nov. 2001.

5. F. Thabtah, P. Cowling and Y. Peng. A New Multiclass,Multi-label Associative Classification Approach. To appear at the 4th International Conference on Data Mining (ICDM ’04). Brighton, UK, Oct. 2004.